**Looting Looters**

**Technical Design Document**

Version 0.9

**Version History**

|  |  |  |
| --- | --- | --- |
| **Version Number** | **Edited By** | **Date** |
| 0.01 | Travis Britton & Dayton Heywood | 20/01/2019 |
| 0.02 | Travis Britton | 21/02/2019 |
| 0.03 | Dayton Heywood | 28/02/2019 |
| 0.04 | Travis Britton | 01/03/2019 |
| 0.9 | Travis Britton & Dayton Heywood | 10/04/2019 |
|  |  |  |
|  |  |  |

**Table of Contents**

[**Game Overview**](#_wo8flfdtjnsy) **3**

[Game Summary](#_30j0zll) 3

[Platform](#_1fob9te) 3

[**Development Overview**](#_3znysh7) **3**

[Development Team](#_2et92p0) 3

[Development Environment](#_tyjcwt) 3

[*Development Hardware*](#_3dy6vkm) *3*

[*Development Software*](#_1t3h5sf) *3*

[*External Code*](#_4d34og8) *3*

[**Game Mechanics**](#_2s8eyo1) **4**

[Main Technical Requirements](#_17dp8vu) 4

[Architecture](#_3rdcrjn) 4

[Game Flow](#_26in1rg) 4

[Graphics](#_lnxbz9) 4

[Audio](#_35nkun2) 4

[Artificial Intelligence](#_ybxg3l4lydjh) 5

[Networking](#_44sinio) 5

[Physics](#_2jxsxqh) 5

[Game Objects and Logic](#_z337ya) 5

[Data Management and Flow](#_3j2qqm3) 7

[**User Interface**](#_1y810tw) **7**

[Game Shell](#_4i7ojhp) 7

[Play Screen](#_2xcytpi) 7

[**Technical Risk**](#_7ghib4ef3nlm) **8**

# 

# Game Overview

## Game Summary

Looting Looters is a competitive first-person online multiplayer game where the goal is to collect as much loot as possible without getting caught by the guard. The winner of a round is determined by whoever scored the most points, not who got caught last... so be aggressive, be smart and LOOT! The players will have access to different traps which can be used to affect other players or the guards as well.

## Platform

PC

# Development Overview

## Development Team

Travis Britton - Programmer

Dayton Heywood - Programmer

~~Valerie Aubut-McWhirter - Artist~~

Raffaele Gambuto - Artist

## Development Environment

### Development Hardware

Different environments were used. Here are some baselines.

OS: Windows 10 Home & Pro

CPU: AMD Ryzen 1800x, Intel i7-8750h

GPU: GTX 1050Ti, GTX 1070

RAM: 8GB+ DDR4

### Development Software

Visual Studio 2017

Unreal Engine 4.20

3DS Max

Photoshop

Illustrator

Git & Github

Google Docs

Dia

Notepad++

### External Code

Unreal Engine Code

Apex Destruction plugin for Unreal Engine

UMG Library for Unreal Engine

GameFrwkSessionsPlugin

# Game Mechanics

## Main Technical Requirements

* Unreal C++ project
* AI, networking, and match-making elements
* Randomly generate level as a collection of rooms
* Populate rooms with randomly selected furniture assets
* Generate room connections via doors randomly connected
* Create traps that can be placed by characters and affect gameplay
* The ability to gather loot which gives score and possibly traps
* Interactable objects which the player can pickup, rotate, drop, throw, etc. When thrown if it impacts something it will destruct (explode into pieces) and then despawn.

## Architecture

See the UML folder for the UML. It does not properly fit here.

See the Pseudocode folder for the Pseudocode.

LEVEL ARCHITECTURE:

Upon game start, the Game randomly generates a level using room blueprints, asset blueprints and loot blueprints. No single runtime of the game will be the same (very unlikely). The level structure of actors is as follow:

GameMode -> Rooms - > Assets -> Loot

The GameMode holds all Rooms, each Room holds all Assets they own, all Assets hold all Loot they own etc.

## Game Flow

Aside from being in the main menu there is no gamestate aside from gameplay unless you get knocked out by the guard which temporarily puts you in a ‘dead’ state.

## Graphics

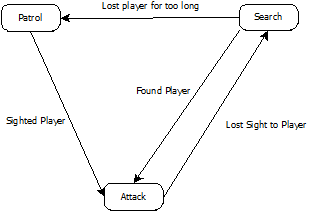
The game runs in 3D using the built in tools of Unreal engine and their constraints on content.

## Audio

We use background music for the main menu and while in the lobby. During gameplay the player has standard background music and when chased that changes to a chase music. There are sounds for picking up loot and traps going off as well.

## 

## Artificial Intelligence



The AI’s main objective is to patrol the map. The AI will enter a room, Find a door in the room and then try to get to it and go through it. Once they go through it this repeats in the new room.

The AI has a PawnSensingComponent with vision turned on.

If he sees the player with this component he will enter attack mode and chase the player. If he contacts the player he will “kill” the player (they respawn after a period).

If he loses sight of the player he enters search mode. In search he stores the player’s last known position and goes there to see if he can find the player. If he still can’t find the player he will try to go through the last door the player went through. If he still can’t find the player after a certain amount of time he will go back to patrol mode. If he does see the player again he re-enters attack mode.

## Networking

Used to replicate most things across server and clients. If one player picks something up everyone sees it, as do they when it shatters. Loot is networked so it’s a race for loot as if one player takes loot other players can’t. Movement, the guard, traps, etc are all networked as well as the players’ inventory and scores.

## Physics

Physics is handled by the default methods inside of Unreal Engine 4.20. Using the basic box, capsule and sphere colliders with Unreal’s physics backend.

## Game Objects and Logic

LEVEL LOADING:

Upon game start, the Game randomly generates a level using room blueprints, asset blueprints and loot blueprints. No single runtime of the game will be the same (very unlikely to get a similar seed). The level structure of actors is as follow:

GameState -> Rooms - > Assets -> Loot

The GameState will then begin generating the level. First the rooms are generated. Then the assets and doors are generated. Then the doors are connected. Finally the loot is spawned. The GameState keeps track of the rooms, which is replicated to all clients to each client has the same level (if this wasn’t server side all players would be playing in their own map).

RANDOMIZATION AND HOW ITS DONE:

There’s numerous levels of randomization to ensure a unique seed on each run. A step by step process of the creation of a game level ensures that:

Step 1 - Generate room meshes randomly from a list of loaded blueprints

Step 2 - Connect the doors of each room together randomly (sometimes there is no connection at all)

Step 3 - Each room spawns their game assets randomly from a list of loaded blueprints using type specifiers

Step 4 - Assets spawn their loot. The loot they spawn is also random.

Step 5 - Loot has a random chance to give traps and also have a varied score based on their type.

AMYPLAYERSTATE

Holds relevant data for the player like their score and team as well as other networked information. Also holds info for the inventory such as how many traps the player has, what slot is selected and what traps are in which slot. Almost exclusively a container.

ABASECHARACTER

All Characters (Guard & Player) inherit from this class. This class sets up most components for characters. It largely handles death, movement and general world interaction. Some functions are overridden in subclasses.

APLAYERCHARACTER

Players are what people actually control. They add a camera on top of the base components. This is where teams are set, music is handled, and player specific actions are handled. Such as respawning or looting. The player can die if they come into contact with the guard. Players on different teams can place traps against other players. The Player also contains an inventory component which handles most UI and trap related interaction. The player’s music changes depending on whether they are being chased or not.

AGUARDCHARACTER

The guards in-game will spawn in their own room. They will travel through the Rooms in the level in an attempt to find a Player. If the guard touches a player they will die and can no longer loot anything until they respawn. If the guard is chasing a player but loses them they will attempt to find that player by using the player’s last known location as well as their last door accessed before losing contact. If at that point they still can’t find the player they go back to patrolling otherwise they attack the player again.

UINVENTORYCOMPONENT

This component is only contained within players. This holds the logic which spawns traps and collects loot. The data is actually held mostly in the playerstate though. When the payer tries to spawn a trap this will check if they have any traps, if the inventory slot selected has a trap in it and if it passes it will spawn whatever trap is in that slot where the player is aiming. When collecting loot it will check if they have any open slots and if the loot gave a trap. If passing it will randomly give the player a trap in the leftmost open slot.

LOOT

Loot is spawned by Assets. The amount of score given is random (clamped to 2 values) as well as a random chance to give a trap. There are multiple tiers of Loot that will have higher chances at getting a trap or guaranteed chances at specific traps. After a certain period of time loot will respawn itself.

TRAPS

Traps are randomly given to the Player when they loot Loot. They cannot affect the actor that placed the trap or that actor’s team. Their effects are always negative and are used as a way to hinder guards and other players. Current traps include:

Stop Trap - The triggering actor is unable to move for a short period of time.

Slow Trap - The triggering actor’s movement speed is slowed for a period of time.

AGRABBABLESTATICMESHACTOR

These actors are the ones which the player can interact with. They can be picked up, rotated, dropped or thrown. When thrown they will basically explode. If thrown at another grabbable they will deal half their health as damage to the other, potentially also breaking that one. These are occasionally spawned in place of standard furniture assets. They can also be used as portable line-of-sight cover from the guard.

ADOORACTOR

These actors represent doors in the game. They have a door mesh and a collider for detecting pawns. When a player collides with the doors collision sphere, they will be teleported to another door. Doors require a pointer to another door to set a ‘connection’. Without a connection the door cannot teleport the player to another door. For that reason, after a level generation if there are any doors leftover that didn’t get a connection (possibly due to an odd number of doors), they are turned invisible and no longer tick or register collision.

AROOMACTORBASE

Rooms are the staple of the level. They hold a mesh that contains the room itself (has a varied amount of floors, walls, ceiling etc). Rooms provide functionality to spawn assets as well as managing their doors. When commanded, the room will iterate through all sockets in its mesh and spawn assets on those sockets using the socket name as a way to determine what asset is spawned there. All spawning is Server side only which is replicated to clients.

Example: SOCKET NAME - Small\_Object\_Chair\_Static\_1

With this name, the room will search the list of asset blueprints for a small chair asset that is not movable.

Rooms are also responsible for connecting doors to other doors on game startup. The GameState calls GenerateDoorConnections, which is a function that iterates through all doors the room is responsible for, and attempts to connect them to other doors using the following rules:

1. The door cannot connect to it’s own room
2. The door cannot connect to a room that is already connected to the owning room through a different door
3. The door cannot connect to itself

During gameplay, the guard character makes use of the rooms to determine patrol points. The guard uses doors as patrol points and also the doors to track player movement when in search mode.

ALOOTINGLOOTERSGAMESTATEBASE

The game state has 3 jobs:

1. Generate the level
2. Start/End the game
3. Hold team data

Job 1: The GameState uses data loaded by the GameMode to generate a random level using Server called functions. The level is then replicated to all clients (this ensures all clients have the same level layout).

Job 2: The GameState has a timer set to limit how long a game lasts. Once that timer is hit, the game is ended and the GameState will tell the GameMode to disconnect the controller inputs from the players via StartEndGame. It will then start a buffer timer to allow players to see their final score. Once that buffer timer is hit, the GameState will call EndMatch on the GameMode, finally ending the match.

Job 3: The GameState keeps bools for determining when players have logged in to the match and also holds a collection of materials for differentiating team colors. When the players are logged in, The GameMode will tell the player to get their team colors from the GameState (which is multicasted so all clients see the updated player color).

ALOOTINGLOOTERSGAMEMODEBASE

The GameMode has 3 jobs:

1. Load game blueprints into memory and hold them.
2. Connect the player/disconnect the player
3. Contain game rules

Job 1: Upon creation, the GameMode uses directories to load all assets, loot, rooms, and doors into memory. These are kept for later as multiple classes will make use of them when calling the Server to spawn objects into the world.

Job 2: When a player connects, the GameMode is responsible for assigning a team to them and then telling them to get their team colors from GameState. When a match is in StartEndGame state, the GameMode will disconnect the players from their input.

Job 3: The GameMode holds how many rooms will spawn in the game. Ideally there would be more rules like additional guards or other things but scope was somewhat reduced over time.

## Data Management and Flow

Due to the nature of the game being basically a random rogue-like nothing is really saved. All assets are loaded on level launch, including the random generation. Nothing gets saved upon exiting the game.

# User Interface

## Game Shell

See Game Design Document - Game Shellfor details.

## Play Screen

See Game Design Document - Play Screen for details.

# 

# Technical Risk

Non-Euclidean Level Design:

Every time the game is launched the level is made up of rooms which are not actually physically attached. Instead the “Doors” are basically just teleporters. Getting this to function reliably and with the AI guards is the biggest problem. The goal is to tackle things one at a time. First have the doors existing as actors, then have them teleport things then act as waypoints for the AI, etc.

Networking:

No one on the team has worked on networking. Through class we are learning how to properly plan and setup the project so the networking can work. It changes several aspects of design and implementation. Knowing what it changes largely resolves this issue.

Random Level Generation:

Every time the game is run the level the player experiences should be relatively different. Different orders to the rooms, different furniture and loot spawned in the rooms, different connections between the rooms, etc. This is a lot of logic to work out but tackling it piece-by-piece limits the difficulty.